Stream Similarity Mining

Erik Vee
Yahoo! Research, Silicon Valley, CA, USA

Synonyms

Distance between streams; Datastream distance

Definition

In many applications, it is useful to think of a datastream as representing a vector or a point in space. Given two datastreams, along with a distance or similarity measure, the distance (or similarity) between the two streams is simply the distance (respectively, similarity) between the two points that the datastreams represent. Due to the enormous amount of data being processed, datastream algorithms are allowed just a single, sequential pass over the data; in some settings, the algorithm may take a few passes. The algorithm itself must use very little memory, typically polylogarithmic in the amount of data, but is allowed to return approximate answers.

There are two frequently used datastream models. In the *time series model*, a vector, $\overrightarrow{x}$, is simply represented as data items arriving in order of their indices: $x_1, x_2, x_3, \ldots$. That is, the value of the ith item of the stream is precisely the value of the ith coordinate of the represented vector. In the *turnstile model*, each arriving item signals an update to some component of the represented vector. So item (i,a) indicates that the value of the ith component of the vector is increased by a. For this reason, datastream items are typically written in the form $(i, x_i^{(j)})$ to indicate that this is the jth update to the ith component of the represented vector. The value of x_i is then the sum of $x_i^{(1)} + x_i^{(2)} + \ldots$ over all such updates. The update values may be negative; the special case when they are restricted to be nonnegative is sometimes called the *cash register model*.

One of the most commonly used measures for datastream similarity is the L_p distance between two streams, for $p \geq 0$. As in the standard definition, the L_p distance between points $\overrightarrow{x}, \overrightarrow{y}$ (hence, between streams representing those points) is defined to be $\sum_i |x_i^p - y_i^p|^{1/p}$. In the case that $p = 0$, the L_0 distance (sometimes called the Hamming distance) is taken to be the number of i such that $x_i \neq y_i$. For $p = \infty$, the L_∞ distance is $\max_i |x_i - y_i|$. Other measures include the Jaccard similarity, the edit distance, the earth-mover's distance, and the length of the longest common subsequence between the streams (viewed as sequences).

Historical Background

Although the earliest datastream-style algorithms were discovered some 30 years ago [11], the current resurgence of interest in datastreams began

© Springer Science+Business Media LLC 2016
L. Liu, M. T. Özsu (eds.), *Encyclopedia of Database Systems*,
DOI 10.1007/978-1-4899-7993-3_373-2

with the seminal paper of Alon et al. [2] in 1996. Implicit in their work is an algorithm for estimating the L_2 distance between streams. In 1999, Feigenbaum et al. [10] developed a datastreaming algorithm to approximate the L_1 distance between two streams. Building on this, Indyk [12] gave datastreaming algorithms to approximate the L_p distance between two datastreams, for all $p \in (0,2]$, utilizing the idea of p-stable distributions. Later, Cormode et al. [7] demonstrated an efficient algorithm for approximating the L_0 distance (i.e., Hamming distance). Sun and Saks [15] provide lower bounds for approximating L_p, for $p > 2$ (and including $p = \infty$), showing no datastream algorithm working in polylogarithmic space can approximate the L_p distance between two streams within a polylogarithmic factor. (The bounds are even stronger for p much larger than 2).

Datar et al. [8] studied the sliding window model for datastreams, producing an algorithm that approximates the L_p distance between two windowed datastreams. Work by Datar and Muthukrishnan [9] gave an algorithm for approximating the Jaccard similarity between two datastreams in the sliding window model.

Foundations

Estimating the L_2 Distance

In their seminal paper, Alon et al. [2] provide a method for estimating F_2, the second frequency moment, of a datastream. As observed in [10, 1], this method can easily be extended to produce a datastream algorithm to approximate the L_2 distance. The ideas are briefly outlined below.

Throughout, the datastreams considered have length n. For $i = 1,2,...,n$, the variable X_i is defined to be an i.i.d. (independent and identically distributed) random variable taking on the value -1 or 1 with equal probability. Of course, a datastream algorithm cannot maintain all the values of each of the random variables in memory. This will be accounted for later; for now, an algorithm is presented assuming that there is random access to these values.

The datastreams vectors are represented in the turnstile model; $(x_1, \ldots, x_n)$ denotes the accumulated values of the first stream, and $(y_1, \ldots, y_n)$ denotes the accumulated values in the second stream. The algorithm simply maintains the value of $\sum_{i=1}^n X_i \cdot \left(x_i - x_y\right)$. This value is straightforward to maintain: If an item $(i, x_i^{(j)})$ arrives for some i, j, the value $X_i \cdot x_i^{(j)}$ is added to it. If an item $(i, y_i^{(j)})$ arrives, the value $X_i \cdot y_i^{(j)}$ is subtracted.

The algorithm focuses on the expected value of *the square* of this quantity:

$$\begin{aligned} &E\left[\left(\sum_{i=1}^n X_i \cdot (x_i - y_i)\right)^2\right] \\ &\quad = E\left[\sum_{i=1}^n X_i^2 \cdot (x_i - y_i)^2 \right. \\ &\quad \left. + \sum_{i \neq j} X_i X_j \cdot (x_i - y_i)\left(x_j - y_j\right)\right] \\ &\quad = \sum_{i=1}^n (x_i - y_i)^2 \end{aligned}$$

where the last equality follows since $E[X_i] = 0$ and $X_i^2 = 1$ for all i, and all the random variables are independent. But this quantity is just the square of the L_2 distance between the two streams. Hence, the problem amounts to obtaining a good estimate of this expected value.

To do so, the above algorithm is run in parallel k times, for $k = \theta(1/\varepsilon^2)$. That is, it maintains the value $\sum_{i=1}^n X_i \cdot \left(x_i - x_y\right)$ for k different random assignments of the X_i. The algorithm then takes the average of their squares. For a given run t, this value is denoted $v^{(t)}$. To further ensure that the algorithm does not obtain a spurious estimate, the procedure is repeated ℓ times, for $\ell = \theta(\log(1/\delta))$. The algorithm then takes the median value over $\{v^{(1)}, v^{(2)}, \ldots, v^{(\ell)}\}$. A standard application of Chebyshev's Inequality shows that this estimates the square of the L_2 distance within a $(1 + \varepsilon)$ factor with probability greater than $1 - \delta$. (In total, this method maintains $k\ell$ values in parallel.)

Unfortunately, the procedure as described above produces and maintains values for n random variables. (In fact, due to the parallel repetitions, it actually needs $k\ell n$ random variables.) However, the technique only needed these variables to be four-wise independent. (Two-wise independence is needed for the expected value to be an unbiased estimator of the square of the L_2 distance; four-wise independence implies that the variance is small.) Hence, these fully independent random variables can be replaced with four-wise independent random variables, which is necessary for Chebyshev's Inequality to hold. These random variables can be pseudorandomly generated on the fly; the datastream algorithm thus only needs to remember a logarithmic-length seed for the pseudorandomly generated values. The full details are omitted here.

Estimating the L_p Distance: p-Stable Distributions

In 2000, Indyk [12], using many of the ideas in [2, 10], extended the results to produce datastream algorithms for approximating the L_p distance between streams, for all $p \in (0,2]$. (Feigenbaum et al. were the first to produce a datastream algorithm for L_1 distance; their technique relied on their construction of pseudorandomly generated "range-summable" variables that were four-wise independent. Although similar in flavor to the result of [2], it is somewhat more complicated). For convenience, the algorithm outlined below details the method for approximating the L_p norm of a single vector. Note, however, that in the turnstile model, it is a simple matter to produce the L_p distance between two streams (by simply negating all of the values in the second stream and finding the norm of their union). Indyk's method uses random linear projections, and relies on the notion of *p-stable distributions*.

A distribution $\mathcal{D}$ is *p-stable* if for all k real numbers $a_1, \ldots, a_k$, if $X_1, \ldots, X_k$ are i.i.d random variables drawn from distribution $\mathcal{D}$, then the random variable $\sum_i a_i X_i$ has the same distribution as $(\sum_i |a_i|^p)^{1/p} X$ for random variable X with distribution $\mathcal{D}$. There are two well-known p-stable distributions. The *Cauchy distribution*, with density function $\mu_C(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable. The *Gaussian distribution*, with density function $\mu_G(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable. Although closed-form functions are not known for p-stable distributions for $p \neq 1,2$, Chambers et al. [4] provide a method for generating p-stable random variables for all $p \in (0,2]$. Throughout the rest of this discussion, $\mathcal{D}$ denotes a p-stable distribution, for some fixed p.

The method for approximating the L_p norm of a stream will now be outlined. As previously noted, this is easily modified to give the L_p distance between two streams. Throughout, the vectors are represented as in the turnstile model, and $(z_1, \ldots, z_n)$ denotes the vector represented by the datastream. As in the previous section, the n i.i.d. random variables $X_1, \ldots, X_n$ are generated first, this time drawn from p-stable distribution $\mathcal{D}$. A brief discussion of how to reduce the number of these variables appears later.

The algorithm simply maintains the value $\sum_i X_i z_i$. Again, these values are easy to maintain: If item $(i, z_i{}^{(j)})$ appears for some i,j, the algorithm adds the value $X_i z_i{}^{(j)}$ to the sum. As in the previous section, the algorithm gains better accuracy by repeating the procedure multiple times in parallel; in this case, the algorithm runs the procedure k times in parallel, for k $= \theta\left(\frac{1}{2}\log(1/\delta)\right)$. The value of $\sum_i X_i z_i$ obtained in the ℓ-th run using this procedure is denoted $Z^{(\ell)}$.

The value $Z^{(\ell)}$ is a random variable itself. Since $\mathcal{D}$ is p-stable, it is the case that $Z^{(\ell)} = X^{(\ell)} \cdot (\sum_i |z_i|^p)^{1/p}$ for some random variable $X^{(\ell)}$ drawn from $\mathcal{D}$. then the output of the algorithm is

$$\frac{1}{\gamma}\text{median}\left\{|Z^{(1)}|, \ldots, |Z^{(k)}|\right\},$$

where γ denotes the median value of $|X|$, for X a random variable distributed according to $\mathcal{D}$. (The absolute value is taken for technical reasons. For instance, the median value of X is 0 when $\mathcal{D}$ is the Gaussian distribution, while the median value of $|X|$ is strictly greater than 0.) The value of the median of $\{|Z^{(1)}|, \ldots, |Z^{(k)}|\}$ is $(\sum_i |z_i|^p)^{1/p}$ times the median of $\{|X^{(1)}|, \ldots, |X^{(k)}|\}$. Hence, the above output is an approximation of $(\sum_i |z_i|^p)^{1/p}$, i.e., the L_p norm of the datastream, as needed. A more careful argument shows that this estimate is within a multiplicative factor $(1 \pm \varepsilon)$ of the true L_p norm, with probability greater than $1 - \delta$.

As in the previous section, Indyk observes that rather than storing the values of n i.i.d random variables, the values can be generated on the fly, using pseudorandom generators. The details are omitted here.

Cormode et al. [7] investigate the problem of estimating the L_0 norm. One of their key technical observations is that the L_p norm is a good approximation of the L_0 norm of the stream, for p sufficiently small. (In particular, they show the $p = \varepsilon / \log M$ is sufficient, where M is the maximum absolute value of any item in the stream.) Thus, the Hamming distance between two streams can be approximated using the same general algorithm that was described above.

Approximating Jaccard Similarity: Min-Wise Hashing

Another useful similarity measure between two streams is their *Jaccard similarity*. Given two datastreams in the time-series model, $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ denote their respective vectors. Further, A (and B) denotes the set of distinct elements appearing in the first stream (respectively, the second stream). The Jaccard similarity between the streams is given by $|A \cap B| / |A \cup B|$.

The first explicit study of the Jaccard similarity between two streams was given by Datar and Muthukrishnan [9]. Their paper examined the sliding window model, which is discussed further in the next section. However, a datastream algorithm in the standard model was given implicitly in the work of Cohen et al. [6], although the notion of datastreams is never mentioned in the paper.

The major technical tool uses *min-wise hashing*, or min-hashing [3, 5]. For every subset A of $[n]$, the *min-hash* for A (with respect to π), denoted h_π (A), is defined to be $h_\pi(A) = \min_{i \in A}\{\pi(i)\}$, where π denotes a permutation on $[n] = \{1, \ldots, n\}$. The wonderful property of the min-hash is that, when π is chosen uniformly at random from the set of all permutations on $[n]$, for any two subsets A,B of $[n]$, it is the case that

$$\Pr[h_\pi(A) = h_\pi(B)] = \frac{|A \cap B|}{|A \cup B|}$$

This suggests the following algorithm.

The algorithm chooses π uniformly at random from the set of permutations on $[n]$. (The fact that storing π take $\theta(n \log n)$ space will be discussed momentarily.) For the first stream, the algorithm finds the value $h_\pi(A) = \min_{i \in A}\{\pi(i)\}$, where A is the set of distinct elements occurring in the first stream. This is simple to do in a datastreaming fashion: as each new a_j appears, the algorithm updates the min value if $\pi(a_j)$is smaller than the min seen so far. Likewise, for the second stream, the algorithm finds the value $h_\pi(B)$, where B is the set of distinct elements occurring in the second stream. From the above, the probability that the two values are equal is precisely the Jaccard similarity between the two streams.

Of course, to obtain an accurate estimate of this probability, the algorithm needs to run the procedure multiple times. In this case, it will run the procedure in parallel k times, each with an independently chosen random permutation. (Here, $k = O(\varepsilon^{-3} \log(1/\delta))$.) The value ρ is defined to be the fraction of times (out of k) that the min values for the two streams coincide. That is, if $\pi_1, \ldots, \pi_k$ are the k independently chosen random permutations, then

$$\rho = \frac{1}{k} \cdot \# \left| \{ j : h_{\pi_j}(A) = h_{\pi_j}(B) \} \right|$$

It is shown in [11] that with probability at least $1 - \delta$, the value ρ approximates the Jaccard similarity within multiplicative factor $(1 \pm \varepsilon)$.

In order for the above algorithm to be useable in a datastreaming context, it must be able to generate and store the necessary random permutations in small space. This is done using *approximately* min-wise independent hash functions. Although this introduces additional error, it can be done in small space and time. The reader is referred to [13] for more details.

Sliding Windows

In many applications, the data from streams becomes outdated or unnecessary quickly. To help understand this scenario better, researchers have proposed the *sliding window* model of datastreams. Here, the algorithm must maintain statistics (e.g., stream similarity), using only the last N

items from the stream, for some N. This causes additional complications, since as each new item comes in, an old item is removed. Since memory is limited, algorithms cannot track which of these old items is disappearing. Still, there are datastream algorithms for both L_p distance and Jaccard similarity in the sliding window model.

In [8], Datar et al. define the sliding window model, and give a datastream algorithm for approximating the L_p distance between two streams (as well as several other datastream algorithms). Their technique uses what they call an *exponential histogram*. The histogram partitions the last N items (i.e., those items in the sliding window) into buckets; the last bucket may in fact contain items older than the last N. Each bucket maintains the necessary statistics for the items it contains. For instance, a bucket containing the items a_s,a_{s+1},...,a_t would hold the L_p -sketch for those items. (Due to memory constraints, the bucket cannot actually maintain the values of all the items it holds.)

As new items come in, the algorithm merges old buckets to maintain the histogram structure, creating new buckets only for newly encountered items. The last bucket will eventually contain only items that do not appear in the N most recent, and will be removed from the histogram at this time. Datar et al. observe that the additional error in this windowed model, beyond that of the standard model, comes from the fact that the last bucket may contain items that are no longer in the N-item window. But the structure of the exponential histogram ensures that this error is not too large. Hence, they provide a general method for translating a wide range of datastream algorithms into windowed-datastream algorithms.

Datar and Muthukrishnan [9] study the problem of approximating the Jaccard similarity of two streams in the sliding window model. As in the non-windowed version, they use min-hashing as a primary tool. The main complication in the sliding window model is that maintaining the minimum value over a sliding window is hard. At a given time step t, the algorithm needs to know the value min $_{i=t,\ldots,t-N+1}\{\pi_j(a_i)\}$, where π_j is a permutation chosen by the datastream algorithm in the standard model. Their solution is to maintain the value $\pi_j(a_i)$for every *relevant*$i = t, \ \ldots \ , t-N+1$. For instance, if $\pi_j(a_i) > \pi_j(a_{i+s})$ for some $s > 0$, then the value $\pi_j(a_i)$will never be the minimum over the sliding window at any time; hence, it may be discarded. (Here, item a_i occurs earlier than a_{i+s}, thus item a_i will move out of the window before a_{i+s}) Amazingly, with high probability, the number of relevant values that need to be maintained is at most $O(\log\ n)$. Hence, the standard datastream algorithm can be adapted to the sliding window model, using small space.

Lower Bounds for Stream Distance

The major technique for proving lower bounds utilize reductions from communication complexity. Here, only sketches of the very high level ideas are presented, with some of the main results cited.

An often-used communication complexity problem is DISJOINTNESS: Alice is given a set, A, and Bob is given a set, B. Neither knows what the other set is. They must communicate with each other by sending messages back and forth, until they decide whether $A \cap B$ is nonempty. (They are allowed to decide ahead of time the protocol they will use to communicate messages.) It has been shown that if the size of A and B is $\theta(n)$, the communication complexity (i.e., the number of bits that must be communicated in the worst case) is also at least $\theta(n)$ [14].

A datastream algorithm that calculates the distance between two streams can provide the basis for a communication complexity algorithm. A typical reduction gives a method for Alice to transform her set A into a datastream (without looking at set B). Likewise, the reduction gives a method for Bob to transform B into a datastream, without looking at A. Finally, the reduction guarantees that Alice's datastream and Bob's datastream are close if and only if $A \cap B$ is non-empty. Then Alice can begin running the datastream algorithm on her datastream. When it has processed her stream, the algorithm will have some memory bits indicating its current state. Alice sends a message to Bob, telling him that state. Bob can then finish running the datastream algorithm on his own datastream. If the algorithm indicates that the two streams are close, he knows $A \cap B$ is nonempty; otherwise, he knows that $A \cap B = \varnothing$ (and may communicate this to Alice in one bit). Hence, Alice and Bob have solved their communication complexity problem. Since the original communication complexity problem took at least $\theta(n)$ bits, the datastream algorithm must also use at least this much memory. (In this case, showing that it cannot be space efficient.)

There is, of course, a great deal of technical work in providing the proper reductions; the difficulties are even greater when showing lower bounds for approximations. However, building on these ideas, Saks and Sun [15] show that approximating the L_∞ distance between two datastreams is impossible to do in sublinear space. In fact, their work shows that approximating within factor $n^{O(\varepsilon)}$ the L_p distance for any $p \geq 2 + \varepsilon$ requires space at least $n^{O(\varepsilon)}$. For p close to 2, this has very little practical implications, but the bounds become more meaningful for large p. Much simpler reductions show the impossibility of space-efficient datastream algorithms for approximating the length of the longest common subsequence between two datastreams (viewed as sequences).

Key Applications

Tracking Change in Network Traffic

The datastream algorithms outlined above allow one to take an entire day of network traffic and synopsize it using a small sketch. It is then possible to measure how different traffic is from day-to-day. Large changes in the network traffic can signal denial of service attacks or worm infestations.

Query Optimization

Most query-optimization techniques utilize data statistics to produce better plans. The L_2 norm is a useful measure for approximating join sizes, while the L_0 norm gives the number of distinct items in the stream.

Processing Genetic Data

Since genetic data consists of millions or billions of base pairs for an individual, it is useful to think of them as streams of data. The similarity of two base-pair sequences is a fundamental concept.

Data Mining

Often individual entities are represented by massive streams of data (e.g., phone calls from a large company, or IP addresses of users visiting a given web site, or items bought at a grocery store). Estimating the similarity between these streams can be a useful tool for identifying similar entities. As one example, it is possible to determine which web sites are most similar to each other, based on the IP addresses of their visitors.

Cross-References

- ▸ Approximation and Data Reduction Techniques
- ▸ Stream Data Management
- ▸ Stream Mining

Recommended Reading

1. Alon N, Gibbons P, Matias Y, Szegedy M. Tracking join and self-join sizes in limited storage. In: Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems. 1999. p. 10–20.
2. Alon N, Matias Y, Szegedy M. The space complexity of approximating the frequency moments. In: Proceedings of the 28th ACM symposium on theory of computing. 1996. p. 20–9.
3. Broder A, Charikar M, Frieze A, Mitzenmacher M. Min-wise independent permutations. In: Proceedings of the 30th ACM symposium on theory of computing. 1998. p. 327–36.
4. Chambers JM, Mallows CL, Stuck BW. A method for simulating stable random variables. J Am Stat Assoc. 1976;71:340–4.
5. Cohen E. Size-estimation framework with applications to transitive closure and reachability. J Comput Syst Sci. 1997;55:441–53.
6. Cohen E, Datar M, Fujiwara S, Gionis A, Indyk P, Motwani R, Ullman J. Finding interesting associations without support pruning. In: Proceedings of the 16th international conference on data engineering. 2000.
7. Cormode G, Datar M, Indyk P, Muthukrishnan S. Comparing data streams using hamming norms. In: Proceedings of the 28th international conference on very large data bases. 2002. p. 335–45.

8. Datar M, Gionis A, Indyk P, Motwani R. Maintaining stream statistics over sliding windows. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms. 2002. p. 635–44.
9. Datar M, Muthukrishnan S. Estimating rarity and similarity on data stream windows. In: Proceedings of the 10th European symposium on algorithms. 2002.
10. Feigenbaum J, Kannan S, Strauss M, Viswanathan M. An approximate l_1-difference algorithm for massive data streams. In: Proceedings of the 40th annual symposium on foundations of computer science. 1999.
11. Flajolet P, Martin G. Probabilistic counting. In: Proceedings of the 24th annual symposium on foundations of computer science. 1983. p. 76–82.
12. Indyk P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In: Proceedings of the 41st annual symposium on foundations of computer science. 2000. p. 189–97.
13. Indyk P. A small approximately min-wise independent family of hash functions. J Algorithm. 2001;38:84–90.
14. On the distributional complexity of disjointness. J Comput Sci Syst. 1984;2.
15. Saks M, Sun X. The space complexity of approximating the frequency moments. In: Proceedings of the 34th ACM symposium on theory of computing. 2002.